INSTRUCTIONS :

1. Complete all questions in your designated project group.
2. All members must contribute to writing the codes. (i.e. 1 question = 1 person, and share the workload if there's an additional question relative to the actual number of members in your team (i.e. 5)). Ensure that all members must understand and explain codes from any of the questions.
3. During viva, all students in each team will be randomly asked to describe, answer and edit any of the answers provided. Marks will be given to your ability to present the answers.

Lab Report

Prepare a report to solve the above problems. The report should contain all the sections as below for each question:

Section	**Description**
1. Problem	Description on the problem
2. Solution	Explanation on how to solve the problems below
3. Sample Input & Output	A few sets of input and output (snapshot)
4. Source code	Java source code

Requirements

1. Group Assignment (Grouping is the same as your project group)
2. Cover page that includes all student matric number and full name.
3. Font: Times New Roman 12, Line Spacing: 1 ½ Spacing
4. Submit to Spectrum according to your OCC. **Deadline** : Before your viva session (W6).

Question 1: The Tok Wan's Number Charms and the Pasar Malam Challenge

Problem Statement:

In the heart of Cheras, not far from the lively Taman Connaught Pasar Malam, lived Tok Wan, a respected elder known for his uncanny ability to create numerous "charms". These weren't for luck, but for a special kind of mental exercise he enjoyed. He would take **3 special numbers: let's call them the "Initial Value" (a), the "Multiplier Seed" (b), and the "Charm Length" (n).**

One evening, after the Isyak prayers, a group of bright young minds from nearby SMK, eager to test problem-solving skills, heard Tok Wan's number charms tales. **They approached him at his warung, with a series of "number queries" (q). First, their leader, Devi, announced a number, (q), indicating the total number of inquiries they had. Following this, for each**

inquiry, they would present Tok Wan with 3 numbers on a single piece of paper: the Initial Value (a), the Multiplier Seed (b), & Charm Length (n).

For each of the "number queries" (q) presented by the bright young minds, your goal is to solve the sequence of numbers that forms Tok Wan's charm. The length of each charm is precisely the "Charm Length" (n) given in the query.

To find each number in the sequence, you must begin with the "Initial Value" (a) provided for that specific query. Then, for each position within the charm, starting from the very first position (which we can think of as the 0th step), you will add a value derived from the "Multiplier Seed" b. The way this "Multiplier Seed" (b) contributes changes with each step in the charm.

- For the very first number in the sequence (at the 0th step), you add the Multiplier Seed multiplied by
  (2^{0}).
- For the next number in the sequence (at the 1st step), you add the Multiplier Seed multiplied by
  (2^{1}). This continues for each subsequent number in the charm.
- For the third number (at the 2nd step), you add
  ($b \times 2^{2}$) (which is ($b \times 4$)), and so on.

You will repeat this process for a total of (n) steps, where (n) is the "Charm Length" given in the query (q). The result of each of these additions will be the numbers that form Tok Wan's charm for that particular inquiry.

If a query gives you an **Initial Value (a) of 1, a Multiplier Seed (b) of 4, and a Charm Length (n) of 4**, the charm would be formed as follows:

- First number (0th step):
  (1+(4×20))=1+(4×1)=1+4=**5**
- Second number (1st step):
  (1+(4×21))=1+(4×2)=1+8=**9**
- Third number (2nd step):
  (1+(4×22))=1+(4×4)=1+16=**17**
- Fourth number (3rd step):
  (1+(4×23))=1+(4×8)=1+32=**33**

So, the sequence for this query would be "**5 9 17 33**". Remember to perform this calculation **for the number of inquiries (q) you receive**. If there are 2 queries, you have to perform 2 iterations for these steps of calculation from the 2 inputs you had.

Input:

```
2           // this is our q
0 2 10       // this is 1st query
5 3 5        // this is 2nd query
```

Output:

For each inquiry (q), you need to provide the **resulting sequence of numbers on a new line**. Within each line, the numbers of the sequence should be **separated by a single space.**

Sample Output:

```
2 4 8 16 32 64 128 256 512 1024     //  output for 1st query
8 11 17 29 53                       //  output for 2nd query
```

Tips:

- The total number of inquiries **(q) will not exceed 500.**
- The **Initial Value (a)** and the **Multiplier Seed (b)** will be non-negative integers **no larger than 50**.
- The Charm Length (n) will be an integer from 1 to 15, **inclusive**.

Question 2: Ah Hock's Digital Signature

Problem Statement:

In the bustling electronic hub of a Cheras Leisure mall, not far from the weekend *pasar malam*, "Ah Hock's Gadget Corner" is legendary. Ah Hock, the young and tech-savvy owner, has a peculiar method for pricing his used gadgets. He believes every serial number or price tag holds a "digital signature" based on its digits. To determine this signature, he uses a daily "Lucky Digit".

His process is interesting. He takes a number, N (like a phone's IMEI or a price), and his chosen "Lucky Digit" for the day, L. He then analyzes N digit by digit, sorting them into categories with a strict order of priority:

1. **Lucky Digits:** First, he counts any digit that matches his "Lucky Digit" L.
2. **Zeroes:** If a digit is not the Lucky Digit, he then checks if it's a zero (0) and counts it.
3. **Even Digits:** If it's neither of the above, he checks if it's an even digit (2, 4, 6, 8) and counts it.
4. **Odd Digits:** If a digit doesn't fall into any of the previous categories, it's counted as an odd digit (1, 3, 5, 7, 9).

After counting, he determines the number's "Digital Signature" based on which category has the highest count.

- If the **Lucky Digit** count is *strictly greater* than all other counts, the signature is LUCKY.
- If the **Even Digit** count is *strictly greater* than all other counts, the signature is BALANCED.
- If the **Odd Digit** count is *strictly greater* than all other counts, the signature is ENERGETIC.
- In any other situation (including any ties for the highest count, or if the Zero count is the highest), the signature is NEUTRAL.

The bright young minds from the local SMK, having mastered Tok Wan's charms, heard about Ah Hock's system and wanted to create a program to predict his signatures.

For each given number N and Lucky Digit L, you must perform Ah Hock's analysis and print the resulting Digital Signature.

- If the number N itself is 0, it is considered to have one digit: 0.
- If the Lucky Digit L is 0, then any 0 in the number N must be counted as a **Lucky Digit**, not as a Zero. The "Zero" category would then only be counted if the Lucky Digit was something other than 0.

Let's say the number to analyze is N=881307 and the Lucky Digit is L=8.

1. **Analyze Digits:** The digits are 8, 8, 1, 3, 0, 7.
2. **Categorize and Count:**
   - Digits matching Lucky Digit 8: 8, 8. **Lucky Count = 2**.
   - Remaining digits: 1, 3, 0, 7.
   - Is any a 0? Yes. **Zero Count = 1**.
   - Remaining digits: 1, 3, 7.
   - Are any even? No. **Even Count = 0**.
   - The rest are odd: 1, 3, 7. **Odd Count = 3**.
3. **Final Counts:** Lucky=2, Zero=1, Even=0, Odd=3.
4. **Determine Signature:** The highest count is 3 (Odd). Since it is strictly greater than all other counts (2, 1, 0), the signature is ENERGETIC.

Another Example (A Tie): If N=2213 and L=5:

- Lucky Count (5): 0
- Zero Count: 0
- Even Count (2): 2
- Odd Count (1,3): 2
- Counts are Lucky=0, Zero=0, Even=2, Odd=2. Since there is a tie for the highest count, the signature is NEUTRAL.

Input:

```
4              // number of queries
881307 8        // test case 1
2213 5          // test case 2
1110 1          // test case 3
8888 8          // test case 4
```

Sample Output:

```
ENERGETIC  // test case 1
NEUTRAL    // test case 2
LUCKY      // test case 3
LUCKY      // test case 4
```

Tips:

- The number of inquiries, T, will be between 1 and 200.
- The number to analyze, N, will be between 0 and 2,000,000,000.
- The Lucky Digit, L, will be a single digit from 0 to 9.

Question 3: Puan Norah's Digital Kolam

Problem Statement:

Down the road from the gadget mall, in a quiet neighbourhood of Cheras, lives Puan Norah, a renowned digital artist. For festive seasons like Deepavali and Hari Raya, she is famous for her beautiful, intricate *kolam* designs which she generates using a program she wrote herself. Her program can create geometric patterns of different sizes and styles.

The students from the local SMK, now inspired by digital creativity, have decided their next challenge is to replicate Puan Norah's pattern generator. They discovered that her program takes two main inputs: a "Height" (H) for the pattern's size, and a "Style" code (S) which determines the pattern's structure.

They have identified two of her primary styles:

- **Style 'A' (Angled):** A simple, right-angled staircase. The number for each row is the row number itself, repeated.
- **Style 'P' (Pyramid):** A more complex, symmetrical pyramid. Each row is a palindrome of numbers counting up from 1 to the row number and then back down to 1. The entire pyramid is neatly centered.

Your program must be able to generate both of Puan Norah's patterns. Based on the given Style code, you will execute the correct logic to print the corresponding pattern for the specified Height.

- **For Style 'A' (Angled):**
    - This is the same as the original "Number Staircase" problem.
    - For a height H, you will print H rows.
    - Row i (where i is from 1 to H) contains the digit i repeated i times.
- **For Style 'P' (Pyramid):**
    - For a height H, you will print H rows.
    - Each row must be padded with leading spaces to be centered. A pyramid of height H has a base width of $(2\times H)-1$.
    - Row i (where i is from 1 to H) will consist of:
        1. A number of leading spaces.
        2. Numbers ascending from 1 up to i.
        3. Numbers descending from $i-1$ back down to 1.

If the input is a Height (H) of 4 and Style (S) of 'P':

- **Row 1 (i=1): 3 spaces + 1 -> 1**
- **Row 2 (i=2): 2 spaces + 12 + 1 -> 121**
- **Row 3 (i=3): 1 space + 123 + 21 -> 12321**
- **Row 4 (i=4): 0 spaces + 1234 + 321 -> 1234321**

Sample Input:

```
3    // number of queries
4 P  // test case 1
5 A  // test case 2
3 P  // test case 3
```

Output:

For each inquiry, provide the resulting pattern. There is no need for extra lines between patterns from different test cases.

Sample Output:

```
  1
  121
 12321
1234321     // end of test case 1
1
22
333
4444
55555        // end of test case 2
  1
 121
12321        // end of test case 3
```

Tips:

- The number of inquiries, T, will be between 1 and 50.
- The Height, H, will be an integer from 1 to 9.
- The Style, S, will be a single uppercase character, either **'A' or 'P'.**

Question 4: The Tale of Tok Wan Osman, Word Gems, and Hidden Values at the Pasar Malam

Problem Statement:

At the lively SS2 Pasar Malam in Petaling Jaya, with the air is full of the delicious smells of satay and nasi lemak, everyone knew Tok Wan. He wasn't famous for selling things, but for his clever word games. He believed that any string of letters had hidden "gems" inside, waiting to be found.

One evening, a group of students from the local SMK school, who often hung out at the market, decided to challenge Tok Wan with a puzzle. They gave him a single long word and a number, which told him the size of the "word gems" to search for.

With a smile, Tok Wan explained that he would look for three special kinds of gems from their word:

1. **"First Whisper"**: He called this the gem that would appear earliest in a dictionary. It is the lexicographically smallest piece of the word.
2. **"Last Echo"**: This is the opposite, the gem that would be found last in a dictionary. It is the lexicographically largest piece.
3. **"Core Value"**: This gem is the "heaviest". He finds it by adding up the **number value (ASCII value)** of each letter in a word piece. The one with the highest total sum is the winner. If two word pieces have the same high value, Tok Wan always picks the first one he finds.

As Tok Wan Osman's grandchildren, inheriting his interest in words, you need to help him decipher the youngsters' puzzles. For each word and gem length given, you must find and name these three word gems, each on a new line.

If the youngsters gave the word (for example) "**satayisverysedap**" and the **gem length 3,** you would need to find:

"First Whisper": The lexicographically smallest substring is **"ata"**.

"Last Echo": The lexicographically largest substring is **"yse"**.

"Core Value": The substring with the highest ASCII sum is **"rys"** (114 + 121 + 115 = 350).

```
"sat": 115 + 97 + 116 = 328

"ata": 97 + 116 + 97 = 310

"tay": 116 + 97 + 121 = 334

"ayi": 97 + 121 + 105 = 323
```

```
"yis": 121 + 105 + 115 = 341

"isv": 105 + 115 + 118 = 338

"sve": 115 + 118 + 101 = 334

"ver": 118 + 101 + 114 = 333

"ery": 101 + 114 + 121 = 336

"rys": 114 + 121 + 115 = 350

"yse": 121 + 115 + 101 = 337

"sed": 115 + 101 + 100 = 316

"eda": 101 + 100 + 97 = 298

"dap": 100 + 97 + 112 = 309
```

Input:

A **single word (string of letters) with no spaces,** followed by a **single positive integer representing the length (k)** of the substrings to analyze.

```
Satayisverysedap    // this is ur test string letter
3                   // this is length (k) of substring
```

Output:

The **"First Whisper",** lexicographically smallest substring of length k found in the input word, then the **"Last Echo",** substring of length k with the highest sum of its characters' ASCII values (the first one encountered in case of a tie) and finally the **"Core Value",** lexicographically largest substring of length k found in the input word.

Sample Output:

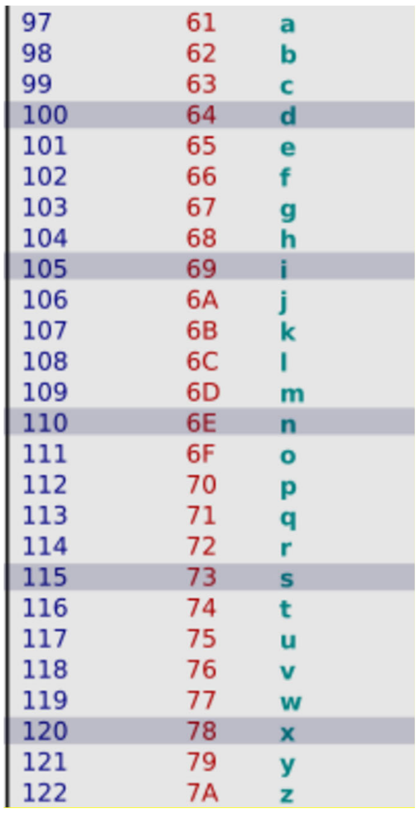
```
ata      // First Whisper
rys      // Last Echo
yse      // Core Value
```

Tips:

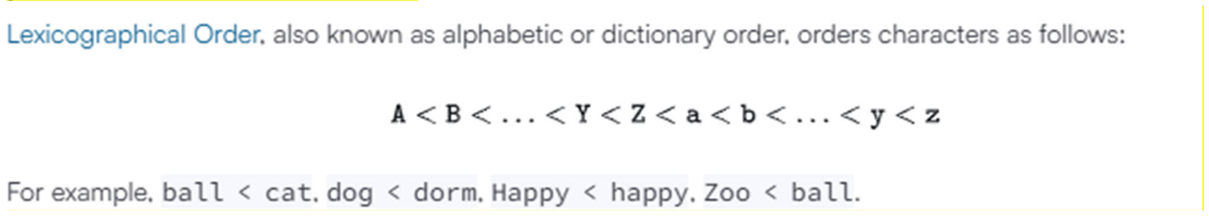
- You will provide a single word, which is a sequence of letters without any spaces.
- **This word will not be longer than 50 characters**.

- The word will only contain English alphabetic letters, and when determining the order for the **"First Whisper"** and **"Last Echo,"** the standard alphabetical comparison used by computers will apply.
- Considering only lowercase. *If uppercase is found, the program should automatically convert it into lowercase.*

Reference:



97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z



Lexicographical Order, also known as alphabetic or dictionary order, orders characters as follows:

$$A < B < \ldots < Y < Z < a < b < \ldots < y < z$$

For example, `ball < cat`, `dog < dorm`, `Happy < happy`, `Zoo < ball`.

Question 5: Uncle Lim's Golden Harmony Lanterns

Problem Statement:

In a corner of the SS2 Pasar Malam, illuminated by a gentle, warm glow, is Uncle Lim's lantern stall. He is a master craftsman, but he is most famous for his "Golden Harmony" collection. These aren't just regular lanterns; the single word painted on each one must follow a strict rule of linguistic harmony he devised himself. A word that breaks the rule is considered "Chaos" and cannot be used.

The students, after tackling word gems and kuih recipes, are now intrigued by Uncle Lim's artistic rules. They want to create a program that can instantly tell if a word possesses "Golden Harmony".

The Rule of Golden Harmony is defined by two simple conditions:

1. A vowel cannot be the very last letter of the word.
2. A vowel cannot be immediately followed by another vowel.

For this puzzle, the vowels are 'a', 'e', 'i', 'o', 'u'. Any other letter is a consonant.

You need to write a validator. For each word given, your program must determine if it follows Uncle Lim's rules for Golden Harmony. If the word follows both rules, it is in harmony. If it breaks even one rule, it is chaos.

Let's analyze the word "meriah":

1. Check the first rule: Does it end in a vowel? No, it ends in 'h'. The rule is not broken.
2. Check the second rule: Are any two vowels adjacent? Let's see.
   - 'e' is a vowel. It is followed by 'r', a consonant. This is okay.
   - 'i' is a vowel. It is followed by 'a', another vowel. This breaks the second rule! Since a rule was broken, the word "meriah" is considered "Chaos".

Let's analyze the word "syukur":

1. Check the first rule: It ends in 'r', a consonant. This is okay.
2. Check the second rule:
   - The first 'u' is followed by 'k' (consonant). Okay.
   - The second 'u' is followed by 'r' (consonant). Okay. Since no rules were broken, the word "syukur" has "Harmony".

Input:

The first line will be a single integer, T, for the number of words to test. The next T lines will each contain a single word to be validated.

```
4 syukur meriah gembira suka
```

Output:

For each word, print "Harmony" or "Chaos" on a new line.

Sample Output:

```
Harmony Chaos Chaos Chaos
```

Tips:

- The number of words to test, T, will be between 1 and 100.
- Each word will consist only of lowercase English alphabetic letters.
- The length of each word will be between 1 and 50 characters.

6) Alex's Stutter Decompression

Problem Statement:

In a brightly lit corner of a bustling Seksyen 13 Digitalmall PJ, just a stone's throw from the weekend market, "Alex's Tech Repair" is the go-to spot for difficult jobs. Alex, an expatriate known for his skill with vintage electronics, has a unique challenge on his hands this Sunday afternoon. He's acquired a collection of rare mobile phones from the early 2000s and is trying to recover their data. The problem is, their error logs are saved in a bizarre, compressed format he's dubbed "Stutter Compression."

The compression scheme saves space by indicating repeated letters with a digit, but not in a simple way. A digit doesn't just represent a count; it's a command to "stutter" the character that came just before it. To make sense of these logs, Alex needs a program to decompress them.

You must write a decompressor for Alex. Your program will read a compressed log file and, if it's valid, calculate the length of the fully decompressed message. However, the format is very fragile, and many logs are corrupted. Your program must also detect these invalid logs.

The Decompression and Validation Rules: You must process the compressed string character by character from left to right.

1. **Letters:** If you read a lowercase letter, it is simply appended to the decompressed message.
2. **Digits ('2'-'9'):** If you read a digit d from '2' to '9', it is a "stutter" command. You must look at the character immediately before the digit in the *original compressed string*. You then append that character to the decompressed message d-1 additional times.

The Rules for an Invalid Log: The process must stop immediately, and the log is declared invalid if any of the following conditions are met:

- **Rule A:** The very first character of the compressed string is a digit. (There's nothing to repeat).
- **Rule B:** A digit is preceded by another digit. (You can't "stutter" a stutter command).
- **Rule C:** The log contains the digits '0' or '1', as they are not used in this compression scheme.

If the log is successfully decompressed without breaking any rules, your output should be the integer length of the final, decompressed string. If any rule is broken, the output must be the exact phrase "Invalid Log".

(Valid Log): Compressed Log: a4b2

1. a: Process 'a'. The decompressed message is now "a".
2. 4: Process '4'. This is a digit. The preceding character in the original string was 'a'. We must repeat 'a' an additional (4-1) = 3 times. The message becomes "aaaa".
3. b: Process 'b'. The message becomes "aaaab".
4. 2: Process '2'. The preceding character was 'b'. Repeat 'b' an additional (2-1) = 1 time. The message becomes "aaaabb". The final decompressed string is "aaaabb". The length is 6. Output: 6

(Invalid Log): Compressed Log: ha33t

1. h: Message is "h".
2. a: Message is "ha".
3. 3: The preceding char was 'a'. Repeat 'a' twice. The message is "haaa".
4. 3: Process '3'. The preceding character in the original string was '3', which is a digit. This breaks Rule B. Stop processing. Output: Invalid Log

Input:

The first line will be a single integer, T, for the number of log strings to test. The next T lines will each contain a single compressed log string.

```
5 a4b2 log5 4bidden test1ng xy22z
```

Output:

For each log string, print either the length of the valid decompressed string or "Invalid Log" on a new line.

Sample Output:

```
6 7 Invalid Log Invalid Log Invalid Log
```

Tips:

- The number of logs, T, will be between 1 and 100.
- The compressed log string will only contain lowercase English letters and digits ('0'-'9').
- The length of the compressed string will be between 1 and 50 characters.
- The final, decompressed string will not exceed 200 characters in length.